

SSC0300 - Linguagens de Programação e Aplicações

Aula01: C/C++ Compilando Programas em Linguagem C - GCC

Professor: Eduardo do Valle Simões

Email: simoesATicmc.usp.br

Objetivo: Aprender a configurar eficientemente o GCC.

1. O Compilador

Para Linux: usa-se o GCC 4.7.1

<http://gcc.gnu.org/>

Para Windows: usa-se o **MinGW** - "Minimalist GNU for Windows", que é o GCC para Windows.

Problema: o **MinGW** usa o GCC-3.4.4-3 !!!

<http://sourceforge.net/projects/mingw/files/>

Solução: **TDM-GCC** usa o GCC 4.6.1 in 32-bit or 64-bit editions!!!

<http://tdm-gcc.tdragon.net/>

Dev-C++ 5.0 beta 9.2 (4.9.9.2) includes full Mingw compiler system with GCC 3.4.2

<http://www.bloodshed.net/>

CODE::BLOCKS 10.05 includes full Mingw compiler system with GCC 3.4.4

<http://www.codeblocks.org>

➔ **Para Windows:** Considerando as versões do GCC, a melhor solução é instalar o TDM-GCC e depois o **CODE::BLOCKS** (versão **SEM** o GCC), pois assim teremos:

- i) Última versão do GCC para Windows
- ii) Uma IDE mais moderna com suporte atualizado.

➔ **Para Linux:** arrume um NERD para lhe ajudar e faça exatamente o que ele mandar!!! Pois quando TUDO der errado (e dará!!!!) ele vai arrumar tudo para ti (ou Morrerá tentando!!!!), será uma questão de honra ao Deus Pinguim!

2. Questões de configuração

Pode-se configurar o GCC passando parâmetros. Pode-se fazer isso diretamente no CMD_Prompt, ou utilizando uma IDE como o CODE::BLOCKS ou o Dev-C++.

Ex.: **gcc** -Wall -Wextra -Wmissing-include-dirs -Wswitch-enum -Wsync-nand -Wfloat-equal -Wlogical-op -Wconversion -Wshadow -Wcast-qual -Wwrite-strings -Wdisabled-optimization -O3 -march=native -funroll-loops **teste.c** -o **teste.exe**

Deve-se configurar os **Warnings** e as opções de **Otimização**.

1) **Warnings**: -Wall → Não liga TODOS os warnings !!!!

Deve-se usar: -Wall -Wextra -Wmissing-include-dirs -Wswitch-enum -Wsync-nand -Wfloat-equal -Wlogical-op -Wconversion -Wshadow -Wcast-qual -Wwrite-strings -Wdisabled-optimization

→ Descrição das diversas opções de Warnings:

<http://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html#Warning-Options>

2) **Optimization**: -O3 -march=native -funroll-loops

-O : Otimiza em 5 níveis:

-O0) Reduz **tempo de compilação** e ajuda o **debug**

-O1) Otimiza para reduzir **tamanho do código** e **tempo de execução**. Sem aumentar **tempo de compilação**.

-O2) Melhora **desempenho**, mais aumenta o **tempo de compilação**.

-O3) Melhora mais ainda o **desempenho**, mais aumenta o **tamanho do código**.

-Os) Otimiza somente o **tamanho do código** em detrimento do **desempenho**.

→ Descrição das diversas opções de Otimização:

<http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html#Optimize-Options>

-march=native : Detecta a máquina alvo (core2, pentium4, pentium4m) e otimiza o código para ela. O código não roda mais em qualquer 386.

-mtune=cpu-type : otimiza um pouco para a máquina alvo, mas permite que o código ainda rode em um 386.

-funroll-loops : desenrola os LOOPS : desmancha os for...

Ex.: Core2 = CPU with 64-bit extensions, MMX, SSE, SSE2, SSE3 and SSSE3 instruction set support

→ Descrição das diversas opções de tuning do Executável:

http://gcc.gnu.org/onlinedocs/gcc/i386-and-x86_002d64-Options.html#i386-and-x86_002d64-Options

Conclusão

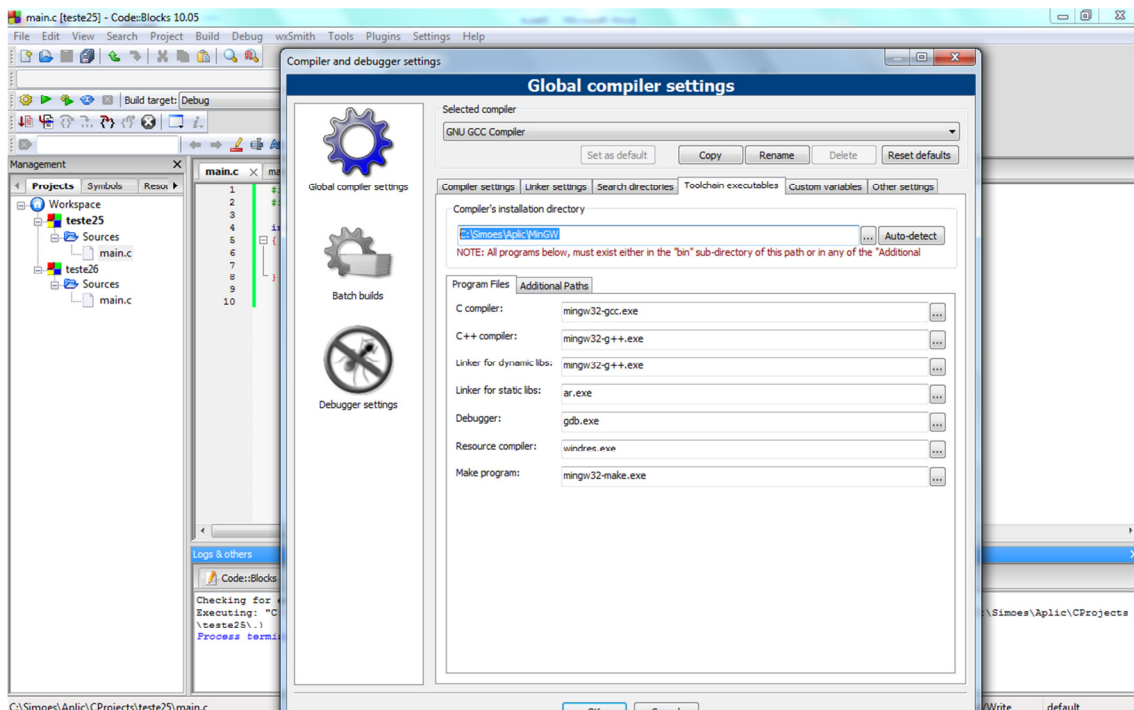
Com muito pouco esforço e conhecimento, é possível melhorar bastante o desempenho das aplicações produzidas. E isso pode ser a diferença entre o software rodar ou não na máquina do cliente.

Para fazer você mesmo (ver sites acima):

- Instale o TDM-GCC
- Instale o code::blocks : Baixar e instalar o code::blocks **versão sem o GCC**: (codeblocks-10.05-setup.exe)
- Ao rodar o code::blocks a primeira vez, ele **deve** achar sozinho o TDMGCC!
- Para testar, inicie um novo projeto tipo “console application” que ele abre um “Hello World”. Compile e rode este programa para testar: **F9**
- Para configurar o compilador no code::blocks, vá em “settings” + “compiler and debugger settings” e insira seus parâmetros na caixa de “options”

OBS.: Se o Code::blocks NÃO DETECTAR O GNU CGG compiler (e aposto que não vai...), faça o seguinte:

- ➔ vá em “settings” + “compiler and debugger settings” e “Toolchain Executables” e insira manualmente o atalho para o MinGW. (o meu está em “C:\Simoes\Aplic\MinGW”).
- ➔ Também verifique se os nomes dos programas estão corretos para C compiler, e os outros, principalmente o C++ e o Linker.



Exemplo 1: Benchmark Program

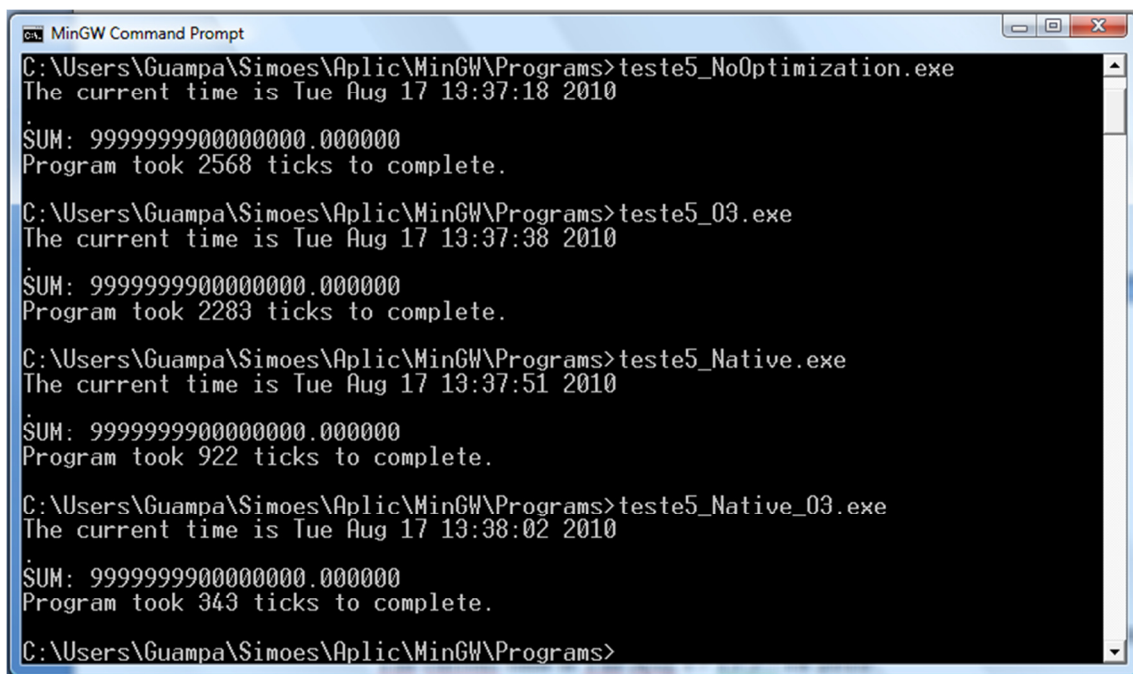
O programa “teste5.c” foi compilado com várias opções de otimização para ilustrar o potencial de otimização de cada variação.

teste5_NoOptimization.exe
Program took 2568 ticks to complete.

teste5_O3.exe
Program took 2283 ticks to complete.

teste5_Native.exe
Program took 922 ticks to complete.

teste5_Native_O3.exe
Program took 343 ticks to complete.



```
MinGW Command Prompt
C:\Users\Guampa\Simoes\Aplic\MinGW\Programs>teste5_NoOptimization.exe
The current time is Tue Aug 17 13:37:18 2010
SUM: 9999999900000000.000000
Program took 2568 ticks to complete.
C:\Users\Guampa\Simoes\Aplic\MinGW\Programs>teste5_O3.exe
The current time is Tue Aug 17 13:37:38 2010
SUM: 9999999900000000.000000
Program took 2283 ticks to complete.
C:\Users\Guampa\Simoes\Aplic\MinGW\Programs>teste5_Native.exe
The current time is Tue Aug 17 13:37:51 2010
SUM: 9999999900000000.000000
Program took 922 ticks to complete.
C:\Users\Guampa\Simoes\Aplic\MinGW\Programs>teste5_Native_O3.exe
The current time is Tue Aug 17 13:38:02 2010
SUM: 9999999900000000.000000
Program took 343 ticks to complete.
C:\Users\Guampa\Simoes\Aplic\MinGW\Programs>
```

Exemplo 2: Compiling a simple C program

The Preprocessor

The Preprocessor accepts source code as input and is responsible for

- * removing comments
- * interpreting special preprocessor directives denoted by #.

For example:

- * #include -- includes contents of a named file (header files).
Ex.: #include <math.h> -- standard library maths file.

`#include <stdio.h> -- standard library I/O file`

* `#define` -- defines a symbolic name or constant. Macro substitution.

Ex.: `#define MAX_ARRAY_SIZE 100`

C Compiler

The C compiler translates source to assembly code. The source code is received from the preprocessor.

Assembler

The assembler creates object code. Files with a `.OBJ` suffix indicate object code files.

Link Editor

If a source file references library function or functions defined in other source files the link editor combines these functions (with `main()`) to create an executable file. External Variable references resolved here also.

Some Useful Compiler Options

-c

Suppress the linking process and produce a `.o` file for each source file listed. Several `.o` files can be subsequently linked by the `cc` command, for example:

`cc file1.o file2.o -o executable`

-llibrary

Link with object libraries. This option must follow the source file arguments. You must link in the libraries explicitly if you wish to use their functions (note do not forget to `#include <library.h>` header file), for example:

`cc calc.c -o calc -lm` \rightarrow `lm = lmath`

Many other libraries are linked in this fashion

-g

invoke debugging option. This instructs the compiler to produce additional symbol table information that is used by a variety of debugging utilities.

-D

define symbols as values (`-Dsymbol=value`) in a similar fashion as the `#define` preprocessor command.

Using Libraries

C is an extremely small language. Many of the functions of other languages are not included in C. e.g. No built in I/O, string handling or maths functions.

All libraries (except standard I/O) need to be explicitly linked in with the -l compiler option described above.

Anexos:

Options to Request or Suppress Warnings

Warnings are diagnostic messages that report constructions that are not inherently erroneous but that are risky or suggest there may have been an error.

The following language-independent options do not enable specific warnings but control the kinds of diagnostics produced by GCC.

`-fsyntax-only`

Check the code for syntax errors, but don't do anything beyond that.

`-fmax-errors=n`

Limits the maximum number of error messages to *n*, at which point GCC bails out rather than attempting to continue processing the source code. If *n* is 0 (the default), there is no limit on the number of error messages produced. If `-Wfatal-errors` is also specified, then `-Wfatal-errors` takes precedence over this option.

`-w`

Inhibit all warning messages.

`-Werror`

Make all warnings into errors.

`-Werror=`

Make the specified warning into an error. The specifier for a warning is appended; for example `-Werror=switch` turns the warnings controlled by `-Wswitch` into errors. This switch takes a negative form, to be used to negate `-Werror` for specific warnings; for example `-Wno-error=switch` makes `-Wswitch` warnings not be errors, even when `-Werror` is in effect.

The warning message for each controllable warning includes the option that controls the warning. That option can then be used with `-Werror=` and `-Wno-error=` as described above. (Printing of the option in the warning message can be disabled using the `-fno-diagnostics-show-option` flag.)

Note that specifying `-Werror=foo` automatically implies `-Wfoo`.

However, `-Wno-error=foo` does not imply anything.

`-Wfatal-errors`

This option causes the compiler to abort compilation on the first error occurred rather than trying to keep going and printing further error messages.

You can request many specific warnings with options beginning with ‘-W’, for example `-Wimplicit` to request warnings on implicit declarations. Each of these specific warning options also has a negative form beginning ‘-Wno-’ to turn off warnings; for example, `-Wno-implicit`. This manual lists only one of the two forms, whichever is not the default. For further language-specific options also refer to [C++ Dialect Options](#) and [Objective-C and Objective-C++ Dialect Options](#).

When an unrecognized warning option is requested (e.g., `-Wunknown-warning`), GCC emits a diagnostic stating that the option is not recognized. However, if the `-Wno-` form is used, the behavior is slightly different: no diagnostic is produced for `-Wno-unknown-warning` unless other diagnostics are being produced. This allows the use of new `-Wno-` options with old compilers, but if something goes wrong, the compiler warns that an unrecognized option is present.

`-Wpedantic`
`-pedantic`

Issue all the warnings demanded by strict ISO C and ISO C++; reject all programs that use forbidden extensions, and some other programs that do not follow ISO C and ISO C++. For ISO C, follows the version of the ISO C standard specified by any `-std` option used.

Valid ISO C and ISO C++ programs should compile properly with or without this option (though a rare few require `-ansi` or a `-std` option specifying the required version of ISO C). However, without this option, certain GNU extensions and traditional C and C++ features are supported as well. With this option, they are rejected.

`-Wpedantic` does not cause warning messages for use of the alternate keywords whose names begin and end with ‘__’. Pedantic warnings are also disabled in the expression that follows `__extension__`.

However, only system header files should use these escape routes; application programs should avoid them. See [Alternate Keywords](#).

Some users try to use `-Wpedantic` to check programs for strict ISO C conformance. They soon find that it does not do quite what they want: it finds some non-ISO practices, but not all—only those for which ISO C *requires* a diagnostic, and some others for which diagnostics have been added.

A feature to report any failure to conform to ISO C might be useful in some instances, but would require considerable additional work and would be quite different from `-Wpedantic`. We don't have plans to support such a feature in the near future.

Where the standard specified with `-std` represents a GNU extended dialect of C, such as ‘gnu90’ or ‘gnu99’, there is a

corresponding *base standard*, the version of ISO C on which the GNU extended dialect is based. Warnings from `-Wpedantic` are given where they are required by the base standard. (It does not make sense for such warnings to be given only for features not in the specified GNU C dialect, since by definition the GNU dialects of C include all features the compiler supports with the given option, and there would be nothing to warn about.)

`-pedantic-errors`

Like `-Wpedantic`, except that errors are produced rather than warnings.

`-Wall`

This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros. This also enables some language-specific warnings described in [C++ Dialect Options](#) and [Objective-C and Objective-C++ Dialect Options](#).

`-Wall` turns on the following warning flags:

- `-Waddress`
- `-Warray-bounds` (only with `-O2`)
- `-Wc++11-compat`
- `-Wchar-subscripts`
- `-Wenum-compare` (in C/ObjC; this is on by default in C++)
- `-Wimplicit-int` (C and Objective-C only)
- `-Wimplicit-function-declaration` (C and Objective-

C only)

- `-Wcomment`
- `-Wformat`
- `-Wmain` (only for C/ObjC and unless `-ffreestanding`)
- `-Wmaybe-uninitialized`
- `-Wmissing-braces` (only for C/ObjC)
- `-Wnonnull`
- `-Wparentheses`
- `-Wpointer-sign`
- `-Wreorder`
- `-Wreturn-type`
- `-Wsequence-point`
- `-Wsign-compare` (only in C++)
- `-Wstrict-aliasing`
- `-Wstrict-overflow=1`
- `-Wswitch`
- `-Wtrigraphs`
- `-Wuninitialized`
- `-Wunknown-pragmas`
- `-Wunused-function`
- `-Wunused-label`
- `-Wunused-value`
- `-Wunused-variable`
- `-Wvolatile-register-var`

Note that some warning flags are not implied by `-Wall`. Some of them warn about constructions that users generally do not consider questionable, but which occasionally you might wish to check for; others warn about constructions that are necessary or hard to avoid in some cases, and there is no simple way to modify the code to suppress the warning. Some of them are enabled by `-Wextra` but many of them must be enabled individually.

`-Wextra`

This enables some extra warning flags that are not enabled by `-Wall`. (This option used to be called `-W`. The older name is still supported, but the newer name is more descriptive.)

- `-Wclobbered`
- `-Wempty-body`
- `-Wignored-qualifiers`
- `-Wmissing-field-initializers`
- `-Wmissing-parameter-type` (C only)
- `-Wold-style-declaration` (C only)
- `-Woverride-init`
- `-Wsign-compare`
- `-Wtype-limits`
- `-Wuninitialized`
- `-Wunused-parameter` (only with `-Wunused` or `-Wall`)
- `-Wunused-but-set-parameter` (only with `-Wunused`

or `-Wall`)

The option `-Wextra` also prints warning messages for the following cases:

- A pointer is compared against integer zero with `'<'`, `'<='`, `'>'`, or `'>='`.
- (C++ only) An enumerator and a non-enumerator both appear in a conditional expression.
- (C++ only) Ambiguous virtual bases.
- (C++ only) Subscripting an array that has been declared `'register'`.
- (C++ only) Taking the address of a variable that has been declared `'register'`.
- (C++ only) A base class is not initialized in a derived class's copy constructor.

`-Wchar-subscripts`

Warn if an array subscript has type `char`. This is a common cause of error, as programmers often forget that this type is signed on some machines. This warning is enabled by `-Wall`.

`-Wcomment`

Warn whenever a comment-start sequence `/*` appears in a `/*` comment, or whenever a Backslash-Newline appears in a `//` comment. This warning is enabled by `-Wall`.

`-Wno-coverage-mismatch`

Warn if feedback profiles do not match when using the `-fprofile-use` option. If a source file is changed between compiling with `-fprofile-gen` and with `-fprofile-use`, the files with the profile feedback can fail to match the source file and GCC cannot use the profile feedback information. By default, this warning is enabled and is treated as an error. `-Wno-coverage-mismatch` can be used to disable the warning or `-Wno-error=coverage-mismatch` can be used to disable the error. Disabling the error for this warning can result in poorly optimized code and is useful only in the case of very minor changes such as bug fixes to an existing code-base. Completely disabling the warning is not recommended.

`-Wno-cpp`

(C, Objective-C, C++, Objective-C++ and Fortran only)

Suppress warning messages emitted by `#warning` directives.

`-Wdouble-promotion` (C, C++, Objective-C and Objective-C++ only)

Give a warning when a value of type `float` is implicitly promoted to `double`. CPUs with a 32-bit “single-precision” floating-point unit implement `float` in hardware, but emulate `double` in software. On such a machine, doing computations using `double` values is much more expensive because of the overhead required for software emulation. It is easy to accidentally do computations with `double` because floating-point literals are implicitly of type `double`. For example, in:

```
float area(float radius)
{
    return 3.14159 * radius * radius;
}
```

the compiler performs the entire computation with `double` because the floating-point literal is a `double`.

`-Wformat`

`-Wformat=n`

Check calls to `printf` and `scanf`, etc., to make sure that the arguments supplied have types appropriate to the format string specified, and that the conversions specified in the format string make sense. This includes standard functions, and others specified by format attributes (see [Function Attributes](#)), in

the `printf`, `scanf`, `strftime` and `strfmon` (an X/Open extension, not in the C standard) families (or other target-specific families). Which functions are checked without format attributes having been specified depends on the standard version selected, and such checks of functions

without the attribute specified are disabled by `-ffreestanding` or `-fno-builtin`.

The formats are checked against the format features supported by GNU libc version 2.2. These include all ISO C90 and C99 features, as well as features from the Single Unix Specification and some BSD and GNU extensions. Other library implementations may not support all these features; GCC does not support warning about features that go beyond a particular library's limitations. However, if `-Wpedantic` is used with `-Wformat`, warnings are given about format features not in the selected standard version (but not for `strfmon` formats, since those are not in any version of the C standard). See [Options Controlling C Dialect](#).

`-Wformat=1`

`-Wformat`

Option `-Wformat` is equivalent to `-Wformat=1`, and `-Wno-format` is equivalent to `-Wformat=0`. Since `-Wformat` also checks for null format arguments for several functions, `-Wformat` also implies `-Wnonnull`. Some aspects of this level of format checking can be disabled by the options: `-Wno-format-contains-nul`, `-Wno-format-extra-args`, and `-Wno-format-zero-length`. `-Wformat` is enabled by `-Wall`.

`-Wno-format-contains-nul`

If `-Wformat` is specified, do not warn about format strings that contain NUL bytes.

`-Wno-format-extra-args`

If `-Wformat` is specified, do not warn about excess arguments to a `printf` or `scanf` format function. The C standard specifies that such arguments are ignored.

Where the unused arguments lie between used arguments that are specified with '\$' operand number specifications, normally warnings are still given, since the implementation could not know what type to pass to `va_arg` to skip the unused arguments. However, in the case of `scanf` formats, this option suppresses the warning if the unused arguments are all pointers, since the Single Unix Specification says that such unused arguments are allowed.

`-Wno-format-zero-length`

If `-Wformat` is specified, do not warn about zero-length formats. The C standard specifies that zero-length formats are allowed.

`-Wformat=2`

Enable `-Wformat` plus additional format checks. Currently equivalent to `-Wformat -Wformat-nonliteral -Wformat-security -Wformat-y2k`.

`-Wformat-nonliteral`

If `-Wformat` is specified, also warn if the format string is not a string literal and so cannot be checked, unless the format function takes its format arguments as a `va_list`.

`-Wformat-security`

If `-Wformat` is specified, also warn about uses of format functions that represent possible security problems. At present, this warns about calls to `printf` and `scanf` functions where the format string is not a string literal and there are no format arguments, as in `printf (foo);`. This may be a security hole if the format string came from untrusted input and contains `'%n'`. (This is currently a subset of what `-Wformat-nonliteral` warns about, but in future warnings may be added to `-Wformat-security` that are not included in `-Wformat-nonliteral`.)

`-Wformat-y2k`

If `-Wformat` is specified, also warn about `strftime` formats that may yield only a two-digit year.

`-Wnonnull`

Warn about passing a null pointer for arguments marked as requiring a non-null value by the `nonnull` function attribute.

`-Wnonnull` is included in `-Wall` and `-Wformat`. It can be disabled with the `-Wno-nonnull` option.

`-Winit-self` (C, C++, Objective-C and Objective-C++ only)

Warn about uninitialized variables that are initialized with themselves. Note this option can only be used with the `-Wuninitialized` option.

For example, GCC warns about `i` being uninitialized in the following snippet only when `-Winit-self` has been specified:

```
int f()
{
    int i = i;
    return i;
}
```

This warning is enabled by `-Wall` in C++.

`-Wimplicit-int` (C and Objective-C only)

Warn when a declaration does not specify a type. This warning is enabled by `-Wall`.

`-Wimplicit-function-declaration` (C and Objective-C only)

Give a warning whenever a function is used before being declared. In C99 mode (`-std=c99` or `-std=gnu99`), this warning is enabled by default and it is made into an error by `-pedantic-errors`. This warning is also enabled by `-Wall`.

`-Wimplicit` (C and Objective-C only)

Same as `-Wimplicit-int` and `-Wimplicit-function-declaration`. This warning is enabled by `-Wall`.

`-Wignored-qualifiers` (C and C++ only)

Warn if the return type of a function has a type qualifier such as `const`. For ISO C such a type qualifier has no effect, since the value returned by a function is not an lvalue. For C++, the warning is only emitted for scalar types or `void`. ISO C prohibits qualified `void` return types on function definitions, so such return types always receive a warning even without this option.

This warning is also enabled by `-Wextra`.

`-Wmain`

Warn if the type of `'main'` is suspicious. `'main'` should be a function with external linkage, returning `int`, taking either zero arguments, two, or three arguments of appropriate types. This warning is enabled by default in C++ and is enabled by either `-Wall` or `-Wpedantic`.

`-Wmissing-braces`

Warn if an aggregate or union initializer is not fully bracketed. In the following example, the initializer for `'a'` is not fully bracketed, but that for `'b'` is fully bracketed. This warning is enabled by `-Wall` in C.

```
int a[2][2] = { 0, 1, 2, 3 };
int b[2][2] = { { 0, 1 }, { 2, 3 } };
```

This warning is enabled by `-Wall`.

`-Wmissing-include-dirs` (C, C++, Objective-C and Objective-C++ only)

Warn if a user-supplied include directory does not exist.

`-Wparentheses`

Warn if parentheses are omitted in certain contexts, such as when there is an assignment in a context where a truth value is expected, or when operators are nested whose precedence people often get confused about. Also warn if a comparison like `'x<=y<=z'` appears; this is equivalent to `'(x<=y ? 1 : 0) <= z'`, which is a different interpretation from that of ordinary mathematical notation.

Also warn about constructions where there may be confusion to which `if` statement an `else` branch belongs. Here is an example of such a case:

```
{
    if (a)
        if (b)
            foo ();
    else
        bar ();
}
```

In C/C++, every `else` branch belongs to the innermost possible `if` statement, which in this example is `if (b)`. This is often not what the programmer expected, as illustrated in the above example by indentation the programmer chose. When there is the potential for this confusion, GCC issues a warning when this flag is specified. To eliminate the warning, add explicit braces around the

innermost `if` statement so there is no way the `else` can belong to the enclosing `if`. The resulting code looks like this:

```
{
    if (a)
    {
        if (b)
            foo ();
        else
            bar ();
    }
}
```

Also warn for dangerous uses of the GNU extension to `?:` with omitted middle operand. When the condition in the `?:` operator is a boolean expression, the omitted value is always 1. Often programmers expect it to be a value computed inside the conditional expression instead.

This warning is enabled by `-Wall`.

`-Wsequence-point`

Warn about code that may have undefined semantics because of violations of sequence point rules in the C and C++ standards. The C and C++ standards define the order in which expressions in a C/C++ program are evaluated in terms of *sequence points*, which represent a partial ordering between the execution of parts of the program: those executed before the sequence point, and those executed after it. These occur after the evaluation of a full expression (one which is not part of a larger expression), after the evaluation of the first operand of a `&&`, `||`, `?:` or `,` (comma) operator, before a function is called (but after the evaluation of its arguments and the expression denoting the called function), and in certain other places. Other than as expressed by the sequence point rules, the order of evaluation of subexpressions of an expression is not specified. All these rules describe only a partial order rather than a total order, since, for example, if two functions are called within one expression with no sequence point between them, the order in which the functions are called is not specified. However, the standards committee have ruled that function calls do not overlap.

It is not specified when between sequence points modifications to the values of objects take effect. Programs whose behavior depends on this have undefined behavior; the C and C++ standards specify that “Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression. Furthermore, the prior value shall be read only to determine the value to be stored.”. If a program breaks these rules, the results on any particular implementation are entirely unpredictable.

Examples of code with undefined behavior are `a = a++;`, `a[n] = b[n++]` and `a[i++] = i;`. Some more complicated cases are not diagnosed by this option, and it may give an occasional false positive

result, but in general it has been found fairly effective at detecting this sort of problem in programs.

The standard is worded confusingly, therefore there is some debate over the precise meaning of the sequence point rules in subtle cases.

Links to discussions of the problem, including proposed formal definitions, may be found on the GCC readings page,

at <http://gcc.gnu.org/readings.html>.

This warning is enabled by `-Wall` for C and C++.

`-Wno-return-local-addr`

Do not warn about returning a pointer (or in C++, a reference) to a variable that goes out of scope after the function returns.

`-Wreturn-type`

Warn whenever a function is defined with a return type that defaults to `int`. Also warn about any `return` statement with no return value in a function whose return type is not `void` (falling off the end of the function body is considered returning without a value), and about a `return` statement with an expression in a function whose return type is `void`.

For C++, a function without return type always produces a diagnostic message, even when `-Wno-return-type` is specified. The only exceptions are ‘main’ and functions defined in system headers.

This warning is enabled by `-Wall`.

`-Wswitch`

Warn whenever a `switch` statement has an index of enumerated type and lacks a `case` for one or more of the named codes of that enumeration. (The presence of a `default` label prevents this warning.) `case` labels outside the enumeration range also provoke warnings when this option is used (even if there is a `default` label).

This warning is enabled by `-Wall`.

`-Wswitch-default`

Warn whenever a `switch` statement does not have a `default` case.

`-Wswitch-enum`

Warn whenever a `switch` statement has an index of enumerated type and lacks a `case` for one or more of the named codes of that enumeration. `case` labels outside the enumeration range also provoke warnings when this option is used. The only difference between `-Wswitch` and this option is that this option gives a warning about an omitted enumeration code even if there is a `default` label.

`-Wsync-nand` (C and C++ only)

Warn when `__sync_fetch_and_nand` and `__sync_nand_and_fetch` built-in functions are used. These functions changed semantics in GCC 4.4.

`-Wtrigraphs`

Warn if any trigraphs are encountered that might change the meaning of the program (trigraphs within comments are not warned about). This warning is enabled by `-Wall`.

`-Wunused-but-set-parameter`

Warn whenever a function parameter is assigned to, but otherwise unused (aside from its declaration).

To suppress this warning use the ‘unused’ attribute (see [Variable Attributes](#)).

This warning is also enabled by `-Wunused` together with `-Wextra`.

`-Wunused-but-set-variable`

Warn whenever a local variable is assigned to, but otherwise unused (aside from its declaration). This warning is enabled by `-Wall`.

To suppress this warning use the ‘unused’ attribute (see [Variable Attributes](#)).

This warning is also enabled by `-Wunused`, which is enabled by `-Wall`.

`-Wunused-function`

Warn whenever a static function is declared but not defined or a non-inline static function is unused. This warning is enabled by `-Wall`.

`-Wunused-label`

Warn whenever a label is declared but not used. This warning is enabled by `-Wall`.

To suppress this warning use the ‘unused’ attribute (see [Variable Attributes](#)).

`-Wunused-local-typedefs` (C, Objective-C, C++ and Objective-C++ only)

Warn when a typedef locally defined in a function is not used. This warning is enabled by `-Wall`.

`-Wunused-parameter`

Warn whenever a function parameter is unused aside from its declaration.

To suppress this warning use the ‘unused’ attribute (see [Variable Attributes](#)).

`-Wno-unused-result`

Do not warn if a caller of a function marked with attribute `warn_unused_result` (see [Function Attributes](#)) does not use its return value. The default is `-Wunused-result`.

`-Wunused-variable`

Warn whenever a local variable or non-constant static variable is unused aside from its declaration. This warning is enabled by `-Wall`.

To suppress this warning use the ‘unused’ attribute (see [Variable Attributes](#)).

`-Wunused-value`

Warn whenever a statement computes a result that is explicitly not used. To suppress this warning cast the unused expression to ‘void’. This includes an expression-statement or the left-hand side of a comma expression that contains no side effects. For example, an expression such as ‘`x[i, j]`’ causes a warning, while ‘`x[(void)i, j]`’ does not.

This warning is enabled by `-Wall`.

`-Wunused`

All the above `-Wunused` options combined.

In order to get a warning about an unused function parameter, you must either specify `-Wextra -Wunused` (note that `-Wall` implies `-Wunused`), or separately specify `-Wunused-parameter`.

`-Wuninitialized`

Warn if an automatic variable is used without first being initialized or if a variable may be clobbered by a `setjmp` call. In C++, warn if a non-static reference or non-static 'const' member appears in a class without constructors.

If you want to warn about code that uses the uninitialized value of the variable in its own initializer, use the `-Winit-self` option.

These warnings occur for individual uninitialized or clobbered elements of structure, union or array variables as well as for variables that are uninitialized or clobbered as a whole. They do not occur for variables or elements declared `volatile`. Because these warnings depend on optimization, the exact variables or elements for which there are warnings depends on the precise optimization options and version of GCC used.

Note that there may be no warning about a variable that is used only to compute a value that itself is never used, because such computations may be deleted by data flow analysis before the warnings are printed.

`-Wmaybe-uninitialized`

For an automatic variable, if there exists a path from the function entry to a use of the variable that is initialized, but there exist some other paths for which the variable is not initialized, the compiler emits a warning if it cannot prove the uninitialized paths are not executed at run time. These warnings are made optional because GCC is not smart enough to see all the reasons why the code might be correct in spite of appearing to have an error. Here is one example of how this can happen:

```
{
    int x;
    switch (y)
    {
        case 1: x = 1;
            break;
        case 2: x = 4;
            break;
        case 3: x = 5;
        }
    foo (x);
}
```

If the value of `y` is always 1, 2 or 3, then `x` is always initialized, but GCC doesn't know this. To suppress the warning, you need to provide a default case with `assert(0)` or similar code.

This option also warns when a non-volatile automatic variable might be changed by a call to `longjmp`. These warnings as well are possible only in optimizing compilation.

The compiler sees only the calls to `setjmp`. It cannot know where `longjmp` will be called; in fact, a signal handler could call it at any point in the code. As a result, you may get a warning even when there is in fact no problem because `longjmp` cannot in fact be called at the place that would cause a problem.

Some spurious warnings can be avoided if you declare all the functions you use that never return as `noreturn`. See [Function Attributes](#).

This warning is enabled by `-Wall` or `-Wextra`.

`-Wunknown-pragmas`

Warn when a `#pragma` directive is encountered that is not understood by GCC. If this command-line option is used, warnings are even issued for unknown pragmas in system header files. This is not the case if the warnings are only enabled by the `-Wall` command-line option.

`-Wno-pragmas`

Do not warn about misuses of pragmas, such as incorrect parameters, invalid syntax, or conflicts between pragmas. See also `-Wunknown-pragmas`.

`-Wstrict-aliasing`

This option is only active when `-fstrict-aliasing` is active. It warns about code that might break the strict aliasing rules that the compiler is using for optimization. The warning does not catch all cases, but does attempt to catch the more common pitfalls. It is included in `-Wall`. It is equivalent to `-Wstrict-aliasing=3`

`-Wstrict-aliasing=n`

This option is only active when `-fstrict-aliasing` is active. It warns about code that might break the strict aliasing rules that the compiler is using for optimization. Higher levels correspond to higher accuracy (fewer false positives). Higher levels also correspond to more effort, similar to the way `-O` works. `-Wstrict-aliasing` is equivalent to `-Wstrict-aliasing=3`.

Level 1: Most aggressive, quick, least accurate. Possibly useful when higher levels do not warn but `-fstrict-aliasing` still breaks the code, as it has very few false negatives. However, it has many false positives. Warns for all pointer conversions between possibly incompatible types, even if never dereferenced. Runs in the front end only.

Level 2: Aggressive, quick, not too precise. May still have many false positives (not as many as level 1 though), and few false negatives (but possibly more than level 1). Unlike level 1, it only warns when an address is taken. Warns about incomplete types. Runs in the front end only.

Level 3 (default for `-Wstrict-aliasing`): Should have very few false positives and few false negatives. Slightly slower than levels 1 or 2 when optimization is enabled. Takes care of the common pun+dereference pattern in the front end: `*(int*)&some_float`. If optimization is enabled, it also runs in the back end, where it deals with multiple statement cases using flow-sensitive points-to information. Only warns when the converted pointer is dereferenced. Does not warn about incomplete types.

`-Wstrict-overflow`

`-Wstrict-overflow=n`

This option is only active when `-fstrict-overflow` is active. It warns about cases where the compiler optimizes based on the assumption that signed overflow does not occur. Note that it does not warn about all cases where the code might overflow: it only warns about cases where the compiler implements some optimization. Thus this warning depends on the optimization level.

An optimization that assumes that signed overflow does not occur is perfectly safe if the values of the variables involved are such that overflow never does, in fact, occur. Therefore this warning can easily give a false positive: a warning about code that is not actually a problem. To help focus on important issues, several warning levels are defined. No warnings are issued for the use of undefined signed overflow when estimating how many iterations a loop requires, in particular when determining whether a loop will be executed at all.

`-Wstrict-overflow=1`

Warn about cases that are both questionable and easy to avoid. For example, with `-fstrict-overflow`, the compiler simplifies `x + 1 > x` to `1`. This level of `-Wstrict-overflow` is enabled by `-Wall`; higher levels are not, and must be explicitly requested.

`-Wstrict-overflow=2`

Also warn about other cases where a comparison is simplified to a constant. For example: `abs (x) >= 0`. This can only be simplified when `-fstrict-overflow` is in effect, because `abs (INT_MIN)` overflows to `INT_MIN`, which is less than zero. `-Wstrict-overflow` (with no level) is the same as `-Wstrict-overflow=2`.

`-Wstrict-overflow=3`

Also warn about other cases where a comparison is simplified. For example: `x + 1 > 1` is simplified to `x > 0`.

`-Wstrict-overflow=4`

Also warn about other simplifications not covered by the above cases. For example: `(x * 10) / 5` is simplified to `x * 2`.

`-Wstrict-overflow=5`

Also warn about cases where the compiler reduces the magnitude of a constant involved in a comparison. For example: `x + 2 > y` is simplified to `x + 1 >= y`. This is reported only at the highest warning

level because this simplification applies to many comparisons, so this warning level gives a very large number of false positives.

`-Wsuggest-attribute=[pure|const|noreturn|format]`

Warn for cases where adding an attribute may be beneficial. The attributes currently supported are listed below.

`-Wsuggest-attribute=pure`

`-Wsuggest-attribute=const`

`-Wsuggest-attribute=noreturn`

Warn about functions that might be candidates for

attributes `pure`, `const` or `noreturn`. The compiler only warns for functions visible in other compilation units or (in the case of `pure` and `const`) if it cannot prove that the function returns normally. A function returns normally if it doesn't contain an infinite loop or return abnormally by throwing, calling `abort()` or trapping. This analysis requires option `-fipa-pure-const`, which is enabled by default at `-O` and higher. Higher optimization levels improve the accuracy of the analysis.

`-Wsuggest-attribute=format`

`-Wmissing-format-attribute`

Warn about function pointers that might be candidates

for `format` attributes. Note these are only possible candidates, not absolute ones. GCC guesses that function pointers with `format` attributes that are used in assignment, initialization, parameter passing or return statements should have a corresponding `format` attribute in the resulting type. I.e. the left-hand side of the assignment or initialization, the type of the parameter variable, or the return type of the containing function respectively should also have a `format` attribute to avoid the warning. GCC also warns about function definitions that might be candidates for `format` attributes. Again, these are only possible candidates. GCC guesses that `format` attributes might be appropriate for any function that calls a function like `vprintf` or `vscanf`, but this might not always be the case, and some functions for which `format` attributes are appropriate may not be detected.

`-Warray-bounds`

This option is only active when `-ftree-vrp` is active (default for `-O2` and above). It warns about subscripts to arrays that are always out of bounds. This warning is enabled by `-Wall`.

`-Wno-div-by-zero`

Do not warn about compile-time integer division by zero. Floating-point division by zero is not warned about, as it can be a legitimate way of obtaining infinities and NaNs.

`-Wsystem-headers`

Print warning messages for constructs found in system header files. Warnings from system headers are normally suppressed, on the

assumption that they usually do not indicate real problems and would only make the compiler output harder to read. Using this command-line option tells GCC to emit warnings from system headers as if they occurred in user code. However, note that using `-Wall` in conjunction with this option does *not* warn about unknown pragmas in system headers—for that, `-Wunknown-pragmas` must also be used.

`-Wtrampolines`

Warn about trampolines generated for pointers to nested functions.

A trampoline is a small piece of data or code that is created at run time on the stack when the address of a nested function is taken, and is used to call the nested function indirectly. For some targets, it is made up of data only and thus requires no special treatment. But, for most targets, it is made up of code and thus requires the stack to be made executable in order for the program to work properly.

`-Wfloat-equal`

Warn if floating-point values are used in equality comparisons.

The idea behind this is that sometimes it is convenient (for the programmer) to consider floating-point values as approximations to infinitely precise real numbers. If you are doing this, then you need to compute (by analyzing the code, or in some other way) the maximum or likely maximum error that the computation introduces, and allow for it when performing comparisons (and when producing output, but that's a different problem). In particular, instead of testing for equality, you should check to see whether the two values have ranges that overlap; and this is done with the relational operators, so equality comparisons are probably mistaken.

`-Wtraditional` (C and Objective-C only)

Warn about certain constructs that behave differently in traditional and ISO C. Also warn about ISO C constructs that have no traditional C equivalent, and/or problematic constructs that should be avoided.

- Macro parameters that appear within string literals in the macro body. In traditional C macro replacement takes place within string literals, but in ISO C it does not.
- In traditional C, some preprocessor directives did not exist. Traditional preprocessors only considered a line to be a directive if the '#' appeared in column 1 on the line. Therefore `-Wtraditional` warns about directives that traditional C understands but ignores because the '#' does not appear as the first character on the line. It also suggests you hide directives like `#pragma` not understood by traditional C by indenting them. Some traditional implementations do not recognize `#elif`, so this option suggests avoiding it altogether.
- A function-like macro that appears without arguments.
- The unary plus operator.

- The ‘U’ integer constant suffix, or the ‘F’ or ‘L’ floating-point constant suffixes. (Traditional C does support the ‘L’ suffix on integer constants.) Note, these suffixes appear in macros defined in the system headers of most modern systems, e.g. the ‘_MIN’/‘_MAX’ macros in `<limits.h>`. Use of these macros in user code might normally lead to spurious warnings, however GCC's integrated preprocessor has enough context to avoid warning in these cases.
- A function declared external in one block and then used after the end of the block.
- A `switch` statement has an operand of type `long`.
- A `non-static` function declaration follows a `static` one. This construct is not accepted by some traditional C compilers.
- The ISO type of an integer constant has a different width or signedness from its traditional type. This warning is only issued if the base of the constant is ten. I.e. hexadecimal or octal values, which typically represent bit patterns, are not warned about.
- Usage of ISO string concatenation is detected.
- Initialization of automatic aggregates.
- Identifier conflicts with labels. Traditional C lacks a separate namespace for labels.
- Initialization of unions. If the initializer is zero, the warning is omitted. This is done under the assumption that the zero initializer in user code appears conditioned on e.g. `__STDC__` to avoid missing initializer warnings and relies on default initialization to zero in the traditional C case.
- Conversions by prototypes between fixed/floating-point values and vice versa. The absence of these prototypes when compiling with traditional C causes serious problems. This is a subset of the possible conversion warnings; for the full set use `-Wtraditional-conversion`.
- Use of ISO C style function definitions. This warning intentionally is *not* issued for prototype declarations or variadic functions because these ISO C features appear in your code when using libiberty's traditional C compatibility macros, `PARAMS` and `VPARAMS`. This warning is also bypassed for nested functions because that feature is already a GCC extension and thus not relevant to traditional C compatibility.

`-Wtraditional-conversion` (C and Objective-C only)

Warn if a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype. This includes conversions of fixed point to floating and vice versa, and

conversions changing the width or signedness of a fixed-point argument except when the same as the default promotion.

`-Wdeclaration-after-statement` (C and Objective-C only)

Warn when a declaration is found after a statement in a block. This construct, known from C++, was introduced with ISO C99 and is by default allowed in GCC. It is not supported by ISO C90 and was not supported by GCC versions before GCC 3.0. See [Mixed Declarations](#).

`-Wundef`

Warn if an undefined identifier is evaluated in an `'#if'` directive.

`-Wno-endif-labels`

Do not warn whenever an `'#else'` or an `'#endif'` are followed by text.

`-Wshadow`

Warn whenever a local variable or type declaration shadows another variable, parameter, type, or class member (in C++), or whenever a built-in function is shadowed. Note that in C++, the compiler warns if a local variable shadows an explicit typedef, but not if it shadows a struct/class/enum.

`-Wlarger-than=len`

Warn whenever an object of larger than *len* bytes is defined.

`-Wframe-larger-than=len`

Warn if the size of a function frame is larger than *len* bytes. The computation done to determine the stack frame size is approximate and not conservative. The actual requirements may be somewhat greater than *len* even if you do not get a warning. In addition, any space allocated via `alloca`, variable-length arrays, or related constructs is not included by the compiler when determining whether or not to issue a warning.

`-Wno-free-nonheap-object`

Do not warn when attempting to free an object that was not allocated on the heap.

`-Wstack-usage=len`

Warn if the stack usage of a function might be larger than *len* bytes. The computation done to determine the stack usage is conservative. Any space allocated via `alloca`, variable-length arrays, or related constructs is included by the compiler when determining whether or not to issue a warning.

The message is in keeping with the output of `-fstack-usage`.

- If the stack usage is fully static but exceeds the specified amount, it's:
 - `warning: stack usage is 1120 bytes`
- If the stack usage is (partly) dynamic but bounded, it's:
 - `warning: stack usage might be 1648 bytes`

- If the stack usage is (partly) dynamic and not bounded, it's:
- `warning: stack usage might be unbounded`

`-Wunsafe-loop-optimizations`

Warn if the loop cannot be optimized because the compiler cannot assume anything on the bounds of the loop indices. With `-funsafe-loop-optimizations` warn if the compiler makes such assumptions.

`-Wno-pedantic-ms-format` (MinGW targets only)

When used in combination with `-Wformat` and `-pedantic` without GNU extensions, this option disables the warnings about non-ISO `printf / scanf` format width specifiers `I32`, `I64`, and `I` used on Windows targets, which depend on the MS runtime.

`-Wpointer-arith`

Warn about anything that depends on the “size of” a function type or of `void`. GNU C assigns these types a size of 1, for convenience in calculations with `void *` pointers and pointers to functions. In C++, warn also when an arithmetic operation involves `NULL`. This warning is also enabled by `-Wpedantic`.

`-Wtype-limits`

Warn if a comparison is always true or always false due to the limited range of the data type, but do not warn for constant expressions. For example, warn if an unsigned variable is compared against zero with `<` or `>=`. This warning is also enabled by `-Wextra`.

`-Wbad-function-cast` (C and Objective-C only)

Warn whenever a function call is cast to a non-matching type. For example, warn if `int malloc()` is cast to `anything *`.

`-Wc++-compat` (C and Objective-C only)

Warn about ISO C constructs that are outside of the common subset of ISO C and ISO C++, e.g. request for implicit conversion from `void *` to a pointer to non-`void` type.

`-Wc++11-compat` (C++ and Objective-C++ only)

Warn about C++ constructs whose meaning differs between ISO C++ 1998 and ISO C++ 2011, e.g., identifiers in ISO C++ 1998 that are keywords in ISO C++ 2011. This warning turns on `-Wnarrowing` and is enabled by `-Wall`.

`-Wcast-qual`

Warn whenever a pointer is cast so as to remove a type qualifier from the target type. For example, warn if a `const char *` is cast to an ordinary `char *`.

Also warn when making a cast that introduces a type qualifier in an unsafe way. For example, casting `char **` to `const char **` is unsafe, as in this example:

```
/* p is char ** value. */
const char **q = (const char **) p;
```



```

/* Assignment of readonly string to const
char * is OK. */
*q = "string";
/* Now char** pointer points to read-only
memory. */
**p = 'b';

```

`-Wcast-align`

Warn whenever a pointer is cast such that the required alignment of the target is increased. For example, warn if a `char *` is cast to an `int *` on machines where integers can only be accessed at two- or four-byte boundaries.

`-Wwrite-strings`

When compiling C, give string constants the type `const char[length]` so that copying the address of one into a non-`const char *` pointer produces a warning. These warnings help you find at compile time code that can try to write into a string constant, but only if you have been very careful about using `const` in declarations and prototypes. Otherwise, it is just a nuisance. This is why we did not make `-Wall` request these warnings.

When compiling C++, warn about the deprecated conversion from string literals to `char *`. This warning is enabled by default for C++ programs.

`-Wclobbered`

Warn for variables that might be changed by ‘`longjmp`’ or ‘`vfork`’. This warning is also enabled by `-Wextra`.

`-Wconditionally-supported` (C++ and Objective-C++ only)

Warn for conditionally-supported (C++11 [intro.defs]) constructs.

`-Wconversion`

Warn for implicit conversions that may alter a value. This includes conversions between real and integer, like `abs (x)` when `x` is double; conversions between signed and unsigned, like `unsigned ui = -1`; and conversions to smaller types, like `sqrtf (M_PI)`. Do not warn for explicit casts like `abs ((int) x)` and `ui = (unsigned) -1`, or if the value is not changed by the conversion like `inabs (2.0)`. Warnings about conversions between signed and unsigned integers can be disabled by using `-Wno-sign-conversion`.

For C++, also warn for confusing overload resolution for user-defined conversions; and conversions that never use a type conversion operator: conversions to `void`, the same type, a base class or a reference to them. Warnings about conversions between signed and unsigned integers are disabled by default in C++ unless `-Wsign-conversion` is explicitly enabled.

`-Wno-conversion-null` (C++ and Objective-C++ only)

Do not warn for conversions between `NULL` and non-pointer types. `-Wconversion-null` is enabled by default.

- Wzero-as-null-pointer-constant (C++ and Objective-C++ only)
Warn when a literal '0' is used as null pointer constant. This can be useful to facilitate the conversion to `nullptr` in C++11.
- Wuseless-cast (C++ and Objective-C++ only)
Warn when an expression is casted to its own type.
- Wempty-body
Warn if an empty body occurs in an 'if', 'else' or 'do while' statement. This warning is also enabled by `-Wextra`.
- Wenum-compare
Warn about a comparison between values of different enumerated types. In C++ enumerals mismatches in conditional expressions are also diagnosed and the warning is enabled by default. In C this warning is enabled by `-Wall`.
- Wjump-misses-init (C, Objective-C only)
Warn if a `goto` statement or a `switch` statement jumps forward across the initialization of a variable, or jumps backward to a label after the variable has been initialized. This only warns about variables that are initialized when they are declared. This warning is only supported for C and Objective-C; in C++ this sort of branch is an error in any case.
-Wjump-misses-init is included in `-Wc++-compat`. It can be disabled with the `-Wno-jump-misses-init` option.
- Wsign-compare
Warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned. This warning is also enabled by `-Wextra`; to get the other warnings of `-Wextra` without this warning, use `-Wextra -Wno-sign-compare`.
- Wsign-conversion
Warn for implicit conversions that may change the sign of an integer value, like assigning a signed integer expression to an unsigned integer variable. An explicit cast silences the warning. In C, this option is enabled also by `-Wconversion`.
- Wsizeof-pointer-memaccess
Warn for suspicious length parameters to certain string and memory built-in functions if the argument uses `sizeof`. This warning warns e.g. about `memset (ptr, 0, sizeof (ptr));` if `ptr` is not an array, but a pointer, and suggests a possible fix, or about `memcpy (&foo, ptr, sizeof (&foo));`. This warning is enabled by `-Wall`.
- Waddress
Warn about suspicious uses of memory addresses. These include using the address of a function in a conditional expression, such as `void func(void); if (func)`, and comparisons against the memory address of a string literal, such as `if (x == "abc")`. Such uses typically indicate a programmer error: the address of a function always evaluates to true, so their use in a conditional usually indicate that the programmer forgot

the parentheses in a function call; and comparisons against string literals result in unspecified behavior and are not portable in C, so they usually indicate that the programmer intended to use `strcmp`. This warning is enabled by `-Wall`.

`-Wlogical-op`

Warn about suspicious uses of logical operators in expressions. This includes using logical operators in contexts where a bit-wise operator is likely to be expected.

`-Waggregate-return`

Warn if any functions that return structures or unions are defined or called. (In languages where you can return an array, this also elicits a warning.)

`-Wno-aggressive-loop-optimizations`

Warn if in a loop with constant number of iterations the compiler detects undefined behavior in some statement during one or more of the iterations.

`-Wno-attributes`

Do not warn if an unexpected `__attribute__` is used, such as unrecognized attributes, function attributes applied to variables, etc. This does not stop errors for incorrect use of supported attributes.

`-Wno-builtin-macro-redefined`

Do not warn if certain built-in macros are redefined. This suppresses warnings for redefinition of `__TIMESTAMP__`, `__TIME__`, `__DATE__`, `__FILE__`, and `__BASE_FILE__`.

`-Wstrict-prototypes` (C and Objective-C only)

Warn if a function is declared or defined without specifying the argument types. (An old-style function definition is permitted without a warning if preceded by a declaration that specifies the argument types.)

`-Wold-style-declaration` (C and Objective-C only)

Warn for obsolescent usages, according to the C Standard, in a declaration. For example, warn if storage-class specifiers like `static` are not the first things in a declaration. This warning is also enabled by `-Wextra`.

`-Wold-style-definition` (C and Objective-C only)

Warn if an old-style function definition is used. A warning is given even if there is a previous prototype.

`-Wmissing-parameter-type` (C and Objective-C only)

A function parameter is declared without a type specifier in K&R-style functions:

```
void foo(bar) { }
```

This warning is also enabled by `-Wextra`.

`-Wmissing-prototypes` (C and Objective-C only)

Warn if a global function is defined without a previous prototype declaration. This warning is issued even if the definition itself provides a prototype. Use this option to detect global functions that do not have a matching prototype declaration in a header file. This option is not valid

for C++ because all function declarations provide prototypes and a non-matching declaration will declare an overload rather than conflict with an earlier declaration. Use `-Wmissing-declarations` to detect missing declarations in C++.

`-Wmissing-declarations`

Warn if a global function is defined without a previous declaration. Do so even if the definition itself provides a prototype. Use this option to detect global functions that are not declared in header files. In C, no warnings are issued for functions with previous non-prototype declarations; use `-Wmissing-prototype` to detect missing prototypes. In C++, no warnings are issued for function templates, or for inline functions, or for functions in anonymous namespaces.

`-Wmissing-field-initializers`

Warn if a structure's initializer has some fields missing. For example, the following code causes such a warning, because `x.h` is implicitly zero:

```
struct s { int f, g, h; };
struct s x = { 3, 4 };
```

This option does not warn about designated initializers, so the following modification does not trigger a warning:

```
struct s { int f, g, h; };
struct s x = { .f = 3, .g = 4 };
```

This warning is included in `-Wextra`. To get other -

`Wextra` warnings without this one, use `-Wextra -Wno-missing-field-initializers`.

`-Wno-multichar`

Do not warn if a multicharacter constant (`'FOOF'`) is used. Usually they indicate a typo in the user's code, as they have implementation-defined values, and should not be used in portable code.

`-Wnormalized=<none|id|nfc|nfkc>`

In ISO C and ISO C++, two identifiers are different if they are different sequences of characters. However, sometimes when characters outside the basic ASCII character set are used, you can have two different character sequences that look the same. To avoid confusion, the ISO 10646 standard sets out some *normalization rules* which when applied ensure that two sequences that look the same are turned into the same sequence. GCC can warn you if you are using identifiers that have not been normalized; this option controls that warning.

There are four levels of warning supported by GCC. The default is `-Wnormalized=nfc`, which warns about any identifier that is not in the ISO 10646 “C” normalized form, *NFC*. *NFC* is the recommended form for most uses.

Unfortunately, there are some characters allowed in identifiers by ISO C and ISO C++ that, when turned into *NFC*, are not allowed in identifiers. That is, there's no way to use these symbols in portable ISO C or C++ and have all your identifiers in *NFC*. -

`Wnormalized=id` suppresses the warning for these characters. It is hoped that future versions of the standards involved will correct this, which is why this option is not the default.

You can switch the warning off for all characters by writing `Wnormalized=none`. You should only do this if you are using some other normalization scheme (like “D”), because otherwise you can easily create bugs that are literally impossible to see.

Some characters in ISO 10646 have distinct meanings but look identical in some fonts or display methodologies, especially once formatting has been applied. For instance `\u207F`, “SUPERSCRIPT LATIN SMALL LETTER N”, displays just like a regular `n` that has been placed in a superscript. ISO 10646 defines the *NFKC* normalization scheme to convert all these into a standard form as well, and GCC warns if your code is not in NFKC if you use `Wnormalized=nfkc`. This warning is comparable to warning about every identifier that contains the letter O because it might be confused with the digit 0, and so is not the default, but may be useful as a local coding convention if the programming environment cannot be fixed to display these characters distinctly.

`-Wno-deprecated`

Do not warn about usage of deprecated features. See [Deprecated Features](#).

`-Wno-deprecated-declarations`

Do not warn about uses of functions (see [Function Attributes](#)), variables (see [Variable Attributes](#)), and types (see [Type Attributes](#)) marked as deprecated by using the `deprecated` attribute.

`-Wno-overflow`

Do not warn about compile-time overflow in constant expressions.

`-Woverride-init` (C and Objective-C only)

Warn if an initialized field without side effects is overridden when using designated initializers (see [Designated Initializers](#)).

This warning is included in `-Wextra`. To get other `-Wextra` warnings without this one, use `-Wextra -Wno-override-init`.

`-Wpacked`

Warn if a structure is given the packed attribute, but the packed attribute has no effect on the layout or size of the structure. Such structures may be mis-aligned for little benefit. For instance, in this code, the variable `f.x` in `struct bar` is misaligned even though `struct bar` does not itself have the packed attribute:

```
struct foo {
    int x;
    char a, b, c, d;
} __attribute__((packed));
struct bar {
    char z;
```

```
    struct foo f;
};
```

`-Wpacked-bitfield-compat`

The 4.1, 4.2 and 4.3 series of GCC ignore the `packed` attribute on bit-fields of type `char`. This has been fixed in GCC 4.4 but the change can lead to differences in the structure layout. GCC informs you when the offset of such a field has changed in GCC 4.4. For example there is no longer a 4-bit padding between field `a` and `b` in this structure:

```
struct foo
{
    char a:4;
    char b:8;
} __attribute__((packed));
```

This warning is enabled by default. Use `-Wno-packed-bitfield-compat` to disable this warning.

`-Wpadded`

Warn if padding is included in a structure, either to align an element of the structure or to align the whole structure. Sometimes when this happens it is possible to rearrange the fields of the structure to reduce the padding and so make the structure smaller.

`-Wredundant-decls`

Warn if anything is declared more than once in the same scope, even in cases where multiple declaration is valid and changes nothing.

`-Wnested-externs` (C and Objective-C only)

Warn if an `extern` declaration is encountered within a function.

`-Wno-inherited-variadic-ctor`

Suppress warnings about use of C++11 inheriting constructors when the base class inherited from has a C variadic constructor; the warning is on by default because the ellipsis is not inherited.

`-Winline`

Warn if a function that is declared as inline cannot be inlined. Even with this option, the compiler does not warn about failures to inline functions declared in system headers.

The compiler uses a variety of heuristics to determine whether or not to inline a function. For example, the compiler takes into account the size of the function being inlined and the amount of inlining that has already been done in the current function. Therefore, seemingly insignificant changes in the source program can cause the warnings produced by `-Winline` to appear or disappear.

`-Wno-invalid-offsetof` (C++ and Objective-C++ only)

Suppress warnings from applying the `'offsetof'` macro to a non-POD type. According to the 1998 ISO C++ standard, applying `'offsetof'` to a non-POD type is undefined. In existing C++ implementations, however, `'offsetof'` typically gives meaningful results even when applied to certain kinds of non-POD types (such as a

simple 'struct' that fails to be a POD type only by virtue of having a constructor). This flag is for users who are aware that they are writing nonportable code and who have deliberately chosen to ignore the warning about it.

The restrictions on 'offsetof' may be relaxed in a future version of the C++ standard.

`-Wno-int-to-pointer-cast`

Suppress warnings from casts to pointer type of an integer of a different size. In C++, casting to a pointer type of smaller size is an error. `Wint-to-pointer-cast` is enabled by default.

`-Wno-pointer-to-int-cast` (C and Objective-C only)

Suppress warnings from casts from a pointer to an integer type of a different size.

`-Winvalid-pch`

Warn if a precompiled header (see [Precompiled Headers](#)) is found in the search path but can't be used.

`-Wlong-long`

Warn if 'long long' type is used. This is enabled by either `-Wpedantic` or `-Wtraditional` in ISO C90 and C++98 modes. To inhibit the warning messages, use `-Wno-long-long`.

`-Wvariadic-macros`

Warn if variadic macros are used in pedantic ISO C90 mode, or the GNU alternate syntax when in pedantic ISO C99 mode. This is default. To inhibit the warning messages, use `-Wno-variadic-macros`.

`-Wvarargs`

Warn upon questionable usage of the macros used to handle variable arguments like 'va_start'. This is default. To inhibit the warning messages, use `-Wno-varargs`.

`-Wvector-operation-performance`

Warn if vector operation is not implemented via SIMD capabilities of the architecture. Mainly useful for the performance tuning. Vector operation can be implemented `piecewise`, which means that the scalar operation is performed on every vector element; `in parallel`, which means that the vector operation is implemented using scalars of wider type, which normally is more performance efficient; and `as a single scalar`, which means that vector fits into a scalar type.

`-Wno-virtual-move-assign`

Suppress warnings about inheriting from a virtual base with a non-trivial C++11 move assignment operator. This is dangerous because if the virtual base is reachable along more than one path, it will be moved multiple times, which can mean both objects end up in the moved-from state. If the move assignment operator is written to avoid moving from a moved-from object, this warning can be disabled.

`-Wvla`

Warn if variable length array is used in the code. `-Wno-vla` prevents the `-Wpedantic` warning of the variable length array.

`-Wvolatile-register-var`

Warn if a register variable is declared volatile. The volatile modifier does not inhibit all optimizations that may eliminate reads and/or writes to register variables. This warning is enabled by `-Wall`.

`-Wdisabled-optimization`

Warn if a requested optimization pass is disabled. This warning does not generally indicate that there is anything wrong with your code; it merely indicates that GCC's optimizers are unable to handle the code effectively. Often, the problem is that your code is too big or too complex; GCC refuses to optimize programs when the optimization itself is likely to take inordinate amounts of time.

`-Wpointer-sign` (C and Objective-C only)

Warn for pointer argument passing or assignment with different signedness. This option is only supported for C and Objective-C. It is implied by `-Wall` and by `-Wpedantic`, which can be disabled with `-Wno-pointer-sign`.

`-Wstack-protector`

This option is only active when `-fstack-protector` is active. It warns about functions that are not protected against stack smashing.

`-Wno-mudflap`

Suppress warnings about constructs that cannot be instrumented by `-fmudflap`.

`-Woverlength-strings`

Warn about string constants that are longer than the “minimum maximum” length specified in the C standard. Modern compilers generally allow string constants that are much longer than the standard's minimum limit, but very portable programs should avoid using longer strings.

The limit applies *after* string constant concatenation, and does not count the trailing NUL. In C90, the limit was 509 characters; in C99, it was raised to 4095. C++98 does not specify a normative minimum maximum, so we do not diagnose overlength strings in C++.

This option is implied by `-Wpedantic`, and can be disabled with `-Wno-overlength-strings`.

`-Wunsuffixed-float-constants` (C and Objective-C only)

Issue a warning for any floating constant that does not have a suffix. When used together with `-Wsystem-headers` it warns about such constants in system header files. This can be useful when preparing code to use with the `FLOAT_CONST_DECIMAL64` pragma from the decimal floating-point extension to C99.

3.17.16 Intel 386 and AMD x86-64 Options

These ‘-m’ options are defined for the i386 and x86-64 family of computers:

`-march=cpu-type`

Generate instructions for the machine type *cpu-type*. In contrast to `-mtune=cpu-type`, which merely tunes the generated code for the specified *cpu-type*, `-march=cpu-type` allows GCC to generate code that may not run at all on processors other than the one indicated.

Specifying `-march=cpu-type` implies `-mtune=cpu-type`.

The choices for *cpu-type* are:

‘native’

This selects the CPU to generate code for at compilation time by determining the processor type of the compiling machine. Using `-march=native` enables all instruction subsets supported by the local machine (hence the result might not run on different machines).

Using `-mtune=native` produces code optimized for the local machine under the constraints of the selected instruction set.

‘i386’

Original Intel i386 CPU.

‘i486’

Intel i486 CPU. (No scheduling is implemented for this chip.)

‘i586’

‘pentium’

Intel Pentium CPU with no MMX support.

‘pentium-mmx’

Intel Pentium MMX CPU, based on Pentium core with MMX instruction set support.

‘pentiumpro’

Intel Pentium Pro CPU.

‘i686’

When used with `-march`, the Pentium Pro instruction set is used, so the code runs on all i686 family chips. When used with `-mtune`, it has the same meaning as ‘generic’.

‘pentium2’

Intel Pentium II CPU, based on Pentium Pro core with MMX instruction set support.

‘pentium3’

‘pentium3m’

Intel Pentium III CPU, based on Pentium Pro core with MMX and SSE instruction set support.

‘pentium-m’

Intel Pentium M; low-power version of Intel Pentium III CPU with MMX, SSE and SSE2 instruction set support. Used by Centrino notebooks.

`'pentium4'`

`'pentium4m'`

Intel Pentium 4 CPU with MMX, SSE and SSE2 instruction set support.

`'prescott'`

Improved version of Intel Pentium 4 CPU with MMX, SSE, SSE2 and SSE3 instruction set support.

`'nocona'`

Improved version of Intel Pentium 4 CPU with 64-bit extensions, MMX, SSE, SSE2 and SSE3 instruction set support.

`'core2'`

Intel Core 2 CPU with 64-bit extensions, MMX, SSE, SSE2, SSE3 and SSSE3 instruction set support.

`'corei7'`

Intel Core i7 CPU with 64-bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1 and SSE4.2 instruction set support.

`'corei7-avx'`

Intel Core i7 CPU with 64-bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, AES and PCLMUL instruction set support.

`'core-avx-i'`

Intel Core CPU with 64-bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, AES, PCLMUL, FSGSBASE, RDRND and F16C instruction set support.

`'core-avx2'`

Intel Core CPU with 64-bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, AVX2, AES, PCLMUL, FSGSBASE, RDRND, FMA, BMI, BMI2 and F16C instruction set support.

`'atom'`

Intel Atom CPU with 64-bit extensions, MOVBE, MMX, SSE, SSE2, SSE3 and SSSE3 instruction set support.

`'slm'`

Intel Silvermont CPU with 64-bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1 and SSE4.2 instruction set support.

`'k6'`

AMD K6 CPU with MMX instruction set support.

`'k6-2'`

`'k6-3'`

Improved versions of AMD K6 CPU with MMX and 3DNow! instruction set support.

`'athlon'`

`'athlon-tbird'`

AMD Athlon CPU with MMX, 3dNOW!, enhanced 3DNow! and SSE prefetch instructions support.

‘athlon-4’

‘athlon-xp’

‘athlon-mp’

Improved AMD Athlon CPU with MMX, 3DNow!, enhanced 3DNow! and full SSE instruction set support.

‘k8’

‘opteron’

‘athlon64’

‘athlon-fx’

Processors based on the AMD K8 core with x86-64 instruction set support, including the AMD Opteron, Athlon 64, and Athlon 64 FX processors. (This supersedes MMX, SSE, SSE2, 3DNow!, enhanced 3DNow! and 64-bit instruction set extensions.)

‘k8-sse3’

‘opteron-sse3’

‘athlon64-sse3’

Improved versions of AMD K8 cores with SSE3 instruction set support.

‘amdfam10’

‘barcelona’

CPUs based on AMD Family 10h cores with x86-64 instruction set support. (This supersedes MMX, SSE, SSE2, SSE3, SSE4A, 3DNow!, enhanced 3DNow!, ABM and 64-bit instruction set extensions.)

‘bdver1’

CPUs based on AMD Family 15h cores with x86-64 instruction set support. (This supersedes FMA4, AVX, XOP, LWP, AES, PCL_MUL, CX16, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM and 64-bit instruction set extensions.)

‘bdver2’

AMD Family 15h core based CPUs with x86-64 instruction set support. (This supersedes BMI, TBM, F16C, FMA, AVX, XOP, LWP, AES, PCL_MUL, CX16, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM and 64-bit instruction set extensions.)

‘bdver3’

AMD Family 15h core based CPUs with x86-64 instruction set support. (This supersedes BMI, TBM, F16C, FMA, AVX, XOP, LWP, AES, PCL_MUL, CX16, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM and 64-bit instruction set extensions.)

‘btver1’

CPUs based on AMD Family 14h cores with x86-64 instruction set support. (This supersedes MMX, SSE, SSE2, SSE3, SSSE3, SSE4A, CX16, ABM and 64-bit instruction set extensions.)

`'btver2'`

CPUs based on AMD Family 16h cores with x86-64 instruction set support. This includes MOVBE, F16C, BMI, AVX, PCL_MUL, AES, SSE4.2, SSE4.1, CX16, ABM, SSE4A, SSSE3, SSE3, SSE2, SSE, MMX and 64-bit instruction set extensions.

`'winchip-c6'`

IDT WinChip C6 CPU, dealt in same way as i486 with additional MMX instruction set support.

`'winchip2'`

IDT WinChip 2 CPU, dealt in same way as i486 with additional MMX and 3DNow! instruction set support.

`'c3'`

VIA C3 CPU with MMX and 3DNow! instruction set support. (No scheduling is implemented for this chip.)

`'c3-2'`

VIA C3-2 (Nehemiah/C5XL) CPU with MMX and SSE instruction set support. (No scheduling is implemented for this chip.)

`'geode'`

AMD Geode embedded processor with MMX and 3DNow! instruction set support.

`-mtune=cpu-type`

Tune to *cpu-type* everything applicable about the generated code, except for the ABI and the set of available instructions. While picking a specific *cpu-type* schedules things appropriately for that particular chip, the compiler does not generate any code that cannot run on the default machine type unless you use a `-march=cpu-type` option. For example, if GCC is configured for i686-pc-linux-gnu then `-mtune=pentium4` generates code that is tuned for Pentium 4 but still runs on i686 machines.

The choices for *cpu-type* are the same as for `-march`. In addition, `-mtune` supports an extra choice for *cpu-type*:

`'generic'`

Produce code optimized for the most common IA32/AMD64/EM64T processors. If you know the CPU on which your code will run, then you should use the corresponding `-mtune` or `-march` option instead of `-mtune=generic`. But, if you do not know exactly what CPU users of your application will have, then you should use this option.

As new processors are deployed in the marketplace, the behavior of this option will change. Therefore, if you upgrade to a newer version of GCC, code generation controlled by this option will change to reflect

the processors that are most common at the time that version of GCC is released.

There is no `-march=generic` option because `-march` indicates the instruction set the compiler can use, and there is no generic instruction set applicable to all processors. In contrast, `-mtune` indicates the processor (or, in this case, collection of processors) for which the code is optimized.

`-mcpu=cpu-type`

A deprecated synonym for `-mtune`.

`-mfpmath=unit`

Generate floating-point arithmetic for selected unit *unit*. The choices for *unit* are:

`'387'`

Use the standard 387 floating-point coprocessor present on the majority of chips and emulated otherwise. Code compiled with this option runs almost everywhere. The temporary results are computed in 80-bit precision instead of the precision specified by the type, resulting in slightly different results compared to most of other chips. See `-ffloat-store` for more detailed description.

This is the default choice for i386 compiler.

`'sse'`

Use scalar floating-point instructions present in the SSE instruction set. This instruction set is supported by Pentium III and newer chips, and in the AMD line by Athlon-4, Athlon XP and Athlon MP chips. The earlier version of the SSE instruction set supports only single-precision arithmetic, thus the double and extended-precision arithmetic are still done using 387. A later version, present only in Pentium 4 and AMD x86-64 chips, supports double-precision arithmetic too.

For the i386 compiler, you must use `-march=cpu-type`, `-msse` or `-msse2` switches to enable SSE extensions and make this option effective. For the x86-64 compiler, these extensions are enabled by default.

The resulting code should be considerably faster in the majority of cases and avoid the numerical instability problems of 387 code, but may break some existing code that expects temporaries to be 80 bits. This is the default choice for the x86-64 compiler.

`'sse, 387'`

`'sse+387'`

`'both'`

Attempt to utilize both instruction sets at once. This effectively doubles the amount of available registers, and on chips with separate execution units for 387 and SSE the execution resources too. Use this option with care, as it is still experimental, because the GCC register allocator does

not model separate functional units well, resulting in unstable performance.

`-masm=`*dialect*

Output assembly instructions using selected *dialect*. Supported choices are 'intel' or 'att' (the default). Darwin does not support 'intel'.

`-mieee-fp`

`-mno-ieee-fp`

Control whether or not the compiler uses IEEE floating-point comparisons. These correctly handle the case where the result of a comparison is unordered.

`-msoft-float`

Generate output containing library calls for floating point.

Warning: the requisite libraries are not part of GCC. Normally the facilities of the machine's usual C compiler are used, but this can't be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

On machines where a function returns floating-point results in the 80387 register stack, some floating-point opcodes may be emitted even if `-msoft-float` is used.

`-mno-fp-ret-in-387`

Do not use the FPU registers for return values of functions.

The usual calling convention has functions return values of types `float` and `double` in an FPU register, even if there is no FPU. The idea is that the operating system should emulate an FPU.

The option `-mno-fp-ret-in-387` causes such values to be returned in ordinary CPU registers instead.

`-mno-fancy-math-387`

Some 387 emulators do not support the `sin`, `cos` and `sqrt` instructions for the 387. Specify this option to avoid generating those instructions.

This option is the default on FreeBSD, OpenBSD and NetBSD. This option is overridden when `-march` indicates that the target CPU always has an FPU and so the instruction does not need emulation.

These instructions are not generated unless you also use the `-funsafe-math-optimizations` switch.

`-malign-double`

`-mno-align-double`

Control whether GCC aligns `double`, `long double`, and `long long` variables on a two-word boundary or a one-word boundary.

Aligning `double` variables on a two-word boundary produces code that runs somewhat faster on a Pentium at the expense of more memory.

On x86-64, `-malign-double` is enabled by default.

Warning: if you use the `-malign-double` switch, structures containing the above types are aligned differently than the published

application binary interface specifications for the 386 and are not binary compatible with structures in code compiled without that switch.

`-m96bit-long-double`
`-m128bit-long-double`

These switches control the size of `long double` type. The i386 application binary interface specifies the size to be 96 bits, so `-m96bit-long-double` is the default in 32-bit mode.

Modern architectures (Pentium and newer) prefer `long double` to be aligned to an 8- or 16-byte boundary. In arrays or structures conforming to the ABI, this is not possible. So specifying `-m128bit-long-double` aligns `long double` to a 16-byte boundary by padding the `long double` with an additional 32-bit zero.

In the x86-64 compiler, `-m128bit-long-double` is the default choice as its ABI specifies that `long double` is aligned on 16-byte boundary.

Notice that neither of these options enable any extra precision over the x87 standard of 80 bits for a `long double`.

Warning: if you override the default value for your target ABI, this changes the size of structures and arrays containing `long double` variables, as well as modifying the function calling convention for functions taking `long double`. Hence they are not binary-compatible with code compiled without that switch.

`-mlong-double-64`
`-mlong-double-80`

These switches control the size of `long double` type. A size of 64 bits makes the `long double` type equivalent to the `double` type. This is the default for Bionic C library.

Warning: if you override the default value for your target ABI, this changes the size of structures and arrays containing `long double` variables, as well as modifying the function calling convention for functions taking `long double`. Hence they are not binary-compatible with code compiled without that switch.

`-mlarge-data-threshold=threshold`

When `-mcmodel=medium` is specified, data objects larger than *threshold* are placed in the large data section. This value must be the same across all objects linked into the binary, and defaults to 65535.

`-mrtd`

Use a different function-calling convention, in which functions that take a fixed number of arguments return with the `ret num` instruction, which pops their arguments while returning. This saves one instruction in the caller since there is no need to pop the arguments there.

You can specify that an individual function is called with this calling sequence with the function attribute `'stdcall'`. You can also override

the `-mrtld` option by using the function attribute `'cdecl'`.

See [Function Attributes](#).

Warning: this calling convention is incompatible with the one normally used on Unix, so you cannot use it if you need to call libraries compiled with the Unix compiler.

Also, you must provide function prototypes for all functions that take variable numbers of arguments (including `printf`); otherwise incorrect code is generated for calls to those functions.

In addition, seriously incorrect code results if you call a function with too many arguments. (Normally, extra arguments are harmlessly ignored.)

`-mregparm=num`

Control how many registers are used to pass integer arguments. By default, no registers are used to pass arguments, and at most 3 registers can be used. You can control this behavior for a specific function by using the function attribute `'regparm'`. See [Function Attributes](#).

Warning: if you use this switch, and *num* is nonzero, then you must build all modules with the same value, including any libraries. This includes the system libraries and startup modules.

`-msseregparm`

Use SSE register passing conventions for float and double arguments and return values. You can control this behavior for a specific function by using the function attribute `'sseregparm'`. See [Function Attributes](#).

Warning: if you use this switch then you must build all modules with the same value, including any libraries. This includes the system libraries and startup modules.

`-mvect8-ret-in-mem`

Return 8-byte vectors in memory instead of MMX registers. This is the default on Solaris 8 and 9 and VxWorks to match the ABI of the Sun Studio compilers until version 12. Later compiler versions (starting with Studio 12 Update 1) follow the ABI used by other x86 targets, which is the default on Solaris 10 and later. *Only* use this option if you need to remain compatible with existing code produced by those previous compiler versions or older versions of GCC.

`-mpc32`

`-mpc64`

`-mpc80`

Set 80387 floating-point precision to 32, 64 or 80 bits. When `-mpc32` is specified, the significands of results of floating-point operations are rounded to 24 bits (single precision); `-mpc64` rounds the significands of results of floating-point operations to 53 bits (double precision) and `-mpc80` rounds the significands of results of floating-point operations to 64 bits (extended double precision), which is the default. When this option is used, floating-point operations in higher

precisions are not available to the programmer without setting the FPU control word explicitly.

Setting the rounding of floating-point operations to less than the default 80 bits can speed some programs by 2% or more. Note that some mathematical libraries assume that extended-precision (80-bit) floating-point operations are enabled by default; routines in such libraries could suffer significant loss of accuracy, typically through so-called “catastrophic cancellation”, when this option is used to set the precision to less than extended precision.

`-mstackrealign`

Realign the stack at entry. On the Intel x86, the `-mstackrealign` option generates an alternate prologue and epilogue that realigns the run-time stack if necessary. This supports mixing legacy codes that keep 4-byte stack alignment with modern codes that keep 16-byte stack alignment for SSE compatibility. See also the attribute `force_align_arg_pointer`, applicable to individual functions.

`-mpreferred-stack-boundary=num`

Attempt to keep the stack boundary aligned to a 2 raised to *num* byte boundary. If `-mpreferred-stack-boundary` is not specified, the default is 4 (16 bytes or 128 bits).

Warning: When generating code for the x86-64 architecture with SSE extensions disabled, `-mpreferred-stack-boundary=3` can be used to keep the stack boundary aligned to 8 byte boundary. Since x86-64 ABI require 16 byte stack alignment, this is ABI incompatible and intended to be used in controlled environment where stack space is important limitation. This option will lead to wrong code when functions compiled with 16 byte stack alignment (such as functions from a standard library) are called with misaligned stack. In this case, SSE instructions may lead to misaligned memory access traps. In addition, variable arguments will be handled incorrectly for 16 byte aligned objects (including x87 long double and `__int128`), leading to wrong results. You must build all modules with `-mpreferred-stack-boundary=3`, including any libraries. This includes the system libraries and startup modules.

`-mincoming-stack-boundary=num`

Assume the incoming stack is aligned to a 2 raised to *num* byte boundary. If `-mincoming-stack-boundary` is not specified, the one specified by `-mpreferred-stack-boundary` is used.

On Pentium and Pentium Pro, `double` and `long double` values should be aligned to an 8-byte boundary (see `-malign-double`) or suffer significant run time performance penalties. On Pentium III, the Streaming SIMD Extension (SSE) data type `__m128` may not work properly if it is not 16-byte aligned.

To ensure proper alignment of this values on the stack, the stack boundary must be as aligned as that required by any value stored on the stack. Further, every function must be generated such that it keeps the stack aligned. Thus calling a function compiled with a higher preferred stack boundary from a function compiled with a lower preferred stack boundary most likely misaligns the stack. It is recommended that libraries that use callbacks always use the default setting.

This extra alignment does consume extra stack space, and generally increases code size. Code that is sensitive to stack space usage, such as embedded systems and operating system kernels, may want to reduce the preferred alignment to `-mpreferred-stack-boundary=2`.

- mmmx
- mno-mmx
- msse
- mno-sse
- msse2
- mno-sse2
- msse3
- mno-sse3
- mssse3
- mno-ssse3
- msse4.1
- mno-sse4.1
- msse4.2
- mno-sse4.2
- msse4
- mno-sse4
- mavx
- mno-avx
- mavx2
- mno-avx2
- maes
- mno-aes
- mpclmul
- mno-pclmul
- mfsgsbase
- mno-fsgsbase
- mrdrnd
- mno-rdrnd
- mf16c
- mno-f16c
- mfma
- mno-fma
- msse4a
- mno-sse4a
- mfma4
- mno-fma4
- mxop
- mno-xop
- mlwp
- mno-lwp
- m3dnow
- mno-3dnow
- mpopcnt
- mno-popcnt
- mabm
- mno-abm

-mbmi
-mbmi2
-mno-bmi
-mno-bmi2
-mlzcnt
-mno-lzcnt
-mrtm
-mtbm
-mno-tbm

These switches enable or disable the use of instructions in the MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, AVX, AVX2, AES, PCLMUL, FSGSBASE, RDRND, F16C, FMA, SSE4A, FMA4, XOP, LWP, ABM, BMI, BMI2, LZCNT, RTM or 3DNow! extended instruction sets. These extensions are also available as built-in functions: see [X86 Built-in Functions](#), for details of the functions enabled and disabled by these switches.

To generate SSE/SSE2 instructions automatically from floating-point code (as opposed to 387 instructions), see `-mfpmath=sse`.

GCC depresses SSEx instructions when `-mavx` is used. Instead, it generates new AVX instructions or AVX equivalence for all SSEx instructions when needed.

These options enable GCC to use these extended instructions in generated code, even without `-mfpmath=sse`. Applications that perform run-time CPU detection must compile separate files for each supported architecture, using the appropriate flags. In particular, the file containing the CPU detection code should be compiled without these options.

-mcld

This option instructs GCC to emit a `cld` instruction in the prologue of functions that use string instructions. String instructions depend on the DF flag to select between autoincrement or autodecrement mode. While the ABI specifies the DF flag to be cleared on function entry, some operating systems violate this specification by not clearing the DF flag in their exception dispatchers. The exception handler can be invoked with the DF flag set, which leads to wrong direction mode when string instructions are used. This option can be enabled by default on 32-bit x86 targets by configuring GCC with the `--enable-cld` configure option. Generation of `cld` instructions can be suppressed with the `-mno-cld` compiler option in this case.

-mvzeroupper

This option instructs GCC to emit a `vzeroupper` instruction before a transfer of control flow out of the function to minimize the AVX to SSE transition penalty as well as remove unnecessary `zeroupper` intrinsics.

-mprefer-avx128

This option instructs GCC to use 128-bit AVX instructions instead of 256-bit AVX instructions in the auto-vectorizer.

-mcx16

This option enables GCC to generate `CMPXCHG16B` instructions. `CMPXCHG16B` allows for atomic operations on 128-bit double quadword (or oword) data types. This is useful for high-resolution counters that can be updated by multiple processors (or cores). This instruction is generated as part of atomic built-in functions: see [__sync Builtins](#) or [__atomic Builtins](#) for details.

`-msahf`

This option enables generation of `SAHF` instructions in 64-bit code. Early Intel Pentium 4 CPUs with Intel 64 support, prior to the introduction of Pentium 4 G1 step in December 2005, lacked the `LAHF` and `SAHF` instructions which were supported by AMD64. These are load and store instructions, respectively, for certain status flags. In 64-bit mode, the `SAHF` instruction is used to optimize `fmod`, `drem`, and `remainder` built-in functions; see [Other Builtins](#) for details.

`-mmovbe`

This option enables use of the `movbe` instruction to implement `__builtin_bswap32` and `__builtin_bswap64`.

`-mcrc32`

This option enables built-in functions `__builtin_ia32_crc32qi`, `__builtin_ia32_crc32hi`, `__builtin_ia32_crc32si` and `__builtin_ia32_crc32di` to generate the `crc32` machine instruction.

`-mrecip`

This option enables use of `RCPPSS` and `RSQRTSS` instructions (and their vectorized variants `RCPPPS` and `RSQRTPS`) with an additional Newton-Raphson step to increase precision instead of `DIVSS` and `SQRTSS` (and their vectorized variants) for single-precision floating-point arguments. These instructions are generated only when `-funsafe-math-optimizations` is enabled together with `-finite-math-only` and `-fno-trapping-math`. Note that while the throughput of the sequence is higher than the throughput of the non-reciprocal instruction, the precision of the sequence can be decreased by up to 2 ulp (i.e. the inverse of 1.0 equals 0.99999994).

Note that GCC implements `1.0f/sqrtf(x)` in terms of `RSQRTSS` (or `RSQRTPS`) already with `-ffast-math` (or the above option combination), and doesn't need `-mrecip`. Also note that GCC emits the above sequence with additional Newton-Raphson step for vectorized single-float division and vectorized `sqrtf(x)` already with `-ffast-math` (or the above option combination), and doesn't need `-mrecip`.

`-mrecip=opt`

This option controls which reciprocal estimate instructions may be used. *opt* is a comma-separated list of options, which may be preceded by a `!` to invert the option:

`'all'`

Enable all estimate instructions.

‘default’

Enable the default instructions, equivalent to `-mrecip`.

‘none’

Disable all estimate instructions, equivalent to `-mno-recip`.

‘div’

Enable the approximation for scalar division.

‘vec-div’

Enable the approximation for vectorized division.

‘sqrt’

Enable the approximation for scalar square root.

‘vec-sqrt’

Enable the approximation for vectorized square root.

So, for example, `-mrecip=all, !sqrt` enables all of the reciprocal approximations, except for square root.

`-mveclibabi=`*type*

Specifies the ABI type to use for vectorizing intrinsics using an external library. Supported values for *type* are ‘svml’ for the Intel short vector math library and ‘acml’ for the AMD math core library.

To use this option, both `-ftree-vectorize` and `-funsafe-math-optimizations` have to be enabled, and an SVML or ACML ABI-compatible library must be specified at link time.

GCC currently emits calls

to `vmldExp2`, `vmldLn2`, `vmldLog102`, `vmldLog102`, `vmldPow2`, `vmldTanh2`, `vmldTan2`, `vmldAtan2`, `vmldAtanh2`, `vmldCbrt2`, `vmldSinh2`, `vmldSin2`, `vmldAsinh2`, `vmldAsin2`, `vmldCosh2`, `vmldCos2`, `vmldAcosh2`, `vmldAcos2`, `vmlsExp4`, `vmlsLn4`, `vmlsLog104`, `vmlsLog104`, `vmlsPow4`, `vmlsTanh4`, `vmlsTan4`, `vmlsAtan4`, `vmlsAtanh4`, `vmlsCbrt4`, `vmlsSinh4`, `vmlsSin4`, `vmlsAsinh4`, `vmlsAsin4`, `vmlsCosh4`, `vmlsCos4`, `vmlsAcosh4` and `vmlsAcos4` for corresponding function type when `-mveclibabi=svml` is used,

and `__vrd2_sin`, `__vrd2_cos`, `__vrd2_exp`, `__vrd2_log`, `__vrd2_log2`, `__vrd2_log10`, `__vrs4_sinf`, `__vrs4_cosf`, `__vrs4_expf`, `__vrs4_logf`, `__vrs4_log2f`, `__vrs4_log10f` and `__vrs4_powf` for the corresponding function type when `-mveclibabi=acml` is used.

`-mabi=`*name*

Generate code for the specified calling convention. Permissible values are ‘sysv’ for the ABI used on GNU/Linux and other systems, and ‘ms’ for the Microsoft ABI. The default is to use the Microsoft ABI when targeting Microsoft Windows and the SysV ABI on all other systems. You can control this behavior for a specific function by using the function attribute ‘ms_abi’/‘sysv_abi’. See [Function Attributes](#).

`-mtls-dialect=`*type*

Generate code to access thread-local storage using the ‘gnu’ or ‘gnu2’ conventions. ‘gnu’ is the conservative default; ‘gnu2’ is more efficient, but it may add compile- and run-time requirements that cannot be satisfied on all systems.

`-mpush-args`

`-mno-push-args`

Use PUSH operations to store outgoing parameters. This method is shorter and usually equally fast as method using SUB/MOV operations and is enabled by default. In some cases disabling it may improve performance because of improved scheduling and reduced dependencies.

`-maccumulate-outgoing-args`

If enabled, the maximum amount of space required for outgoing arguments is computed in the function prologue. This is faster on most modern CPUs because of reduced dependencies, improved scheduling and reduced stack usage when the preferred stack boundary is not equal to 2. The drawback is a notable increase in code size. This switch implies `-mno-push-args`.

`-mthreads`

Support thread-safe exception handling on MinGW. Programs that rely on thread-safe exception handling must compile and link all code with the `-mthreads` option. When compiling, `-mthreads` defines `_D_MT`; when linking, it links in a special thread helper library – `lmingwthrd` which cleans up per-thread exception-handling data.

`-mno-align-stringops`

Do not align the destination of inlined string operations. This switch reduces code size and improves performance in case the destination is already aligned, but GCC doesn't know about it.

`-minline-all-stringops`

By default GCC inlines string operations only when the destination is known to be aligned to least a 4-byte boundary. This enables more inlining and increases code size, but may improve performance of code that depends on fast `memcpy`, `strlen`, and `memset` for short lengths.

`-minline-stringops-dynamically`

For string operations of unknown size, use run-time checks with inline code for small blocks and a library call for large blocks.

`-mstringop-strategy=alg`

Override the internal decision heuristic for the particular algorithm to use for inlining string operations. The allowed values for *alg* are:

‘rep_byte’

‘rep_4byte’

‘rep_8byte’

Expand using `i386_rep` prefix of the specified size.

‘byte_loop’

‘loop’

‘unrolled_loop’

Expand into an inline loop.

`'libcall'`

Always use a library call.

`-fomit-leaf-frame-pointer`

Don't keep the frame pointer in a register for leaf functions. This avoids the instructions to save, set up, and restore frame pointers and makes an extra register available in leaf functions. The option `-fomit-leaf-frame-pointer` removes the frame pointer for leaf functions, which might make debugging harder.

`-mtls-direct-seg-refs`

`-mno-tls-direct-seg-refs`

Controls whether TLS variables may be accessed with offsets from the TLS segment register (`%gs` for 32-bit, `%fs` for 64-bit), or whether the thread base pointer must be added. Whether or not this is valid depends on the operating system, and whether it maps the segment to cover the entire TLS area.

For systems that use the GNU C Library, the default is on.

`-msse2avx`

`-mno-sse2avx`

Specify that the assembler should encode SSE instructions with VEX prefix. The option `-mavx` turns this on by default.

`-mfentry`

`-mno-fentry`

If profiling is active (`-pg`), put the profiling counter call before the prologue. Note: On x86 architectures the attribute `ms_hook_prologue` isn't possible at the moment for `-mfentry` and `-pg`.

`-m8bit-idiv`

`-mno-8bit-idiv`

On some processors, like Intel Atom, 8-bit unsigned integer divide is much faster than 32-bit/64-bit integer divide. This option generates a run-time check. If both dividend and divisor are within range of 0 to 255, 8-bit unsigned integer divide is used instead of 32-bit/64-bit integer divide.

`-mavx256-split-unaligned-load`

`-mavx256-split-unaligned-store`

Split 32-byte AVX unaligned load and store.

`-mstack-protector-guard=guard`

Generate stack protection code using canary at *guard*. Supported locations are `'global'` for global canary or `'tls'` for per-thread canary in the TLS block (the default). This option has effect only when `-fstack-protector` or `-fstack-protector-all` is specified.

These `'-m'` switches are supported in addition to the above on x86-64 processors in 64-bit environments.

`-m32`

`-m64`
`-mx32`

Generate code for a 32-bit or 64-bit environment. The `-m32` option sets `int`, `long`, and pointer types to 32 bits, and generates code that runs on any i386 system.

The `-m64` option sets `int` to 32 bits and `long` and pointer types to 64 bits, and generates code for the x86-64 architecture. For Darwin only the `-m64` option also turns off the `-fno-pic` and `-mdynamic-no-pic` options.

The `-mx32` option sets `int`, `long`, and pointer types to 32 bits, and generates code for the x86-64 architecture.

`-mno-red-zone`

Do not use a so-called “red zone” for x86-64 code. The red zone is mandated by the x86-64 ABI; it is a 128-byte area beyond the location of the stack pointer that is not modified by signal or interrupt handlers and therefore can be used for temporary data without adjusting the stack pointer. The flag `-mno-red-zone` disables this red zone.

`-mcmodel=small`

Generate code for the small code model: the program and its symbols must be linked in the lower 2 GB of the address space. Pointers are 64 bits. Programs can be statically or dynamically linked. This is the default code model.

`-mcmodel=kernel`

Generate code for the kernel code model. The kernel runs in the negative 2 GB of the address space. This model has to be used for Linux kernel code.

`-mcmodel=medium`

Generate code for the medium model: the program is linked in the lower 2 GB of the address space. Small symbols are also placed there. Symbols with sizes larger than `-mlarge-data-threshold` are put into large data or BSS sections and can be located above 2GB. Programs can be statically or dynamically linked.

`-mcmodel=large`

Generate code for the large model. This model makes no assumptions about addresses and sizes of sections.

`-maddress-mode=long`

Generate code for long address mode. This is only supported for 64-bit and x32 environments. It is the default address mode for 64-bit environments.

`-maddress-mode=short`

Generate code for short address mode. This is only supported for 32-bit and x32 environments. It is the default address mode for 32-bit and x32 environments.